

Type Derivations

Mattox Beckman

`beckman@iit.edu`

Illinois Institute of Technology

In order to express the meaning of a program, we need a formal language to capture these meanings. One way to express meaning is to say something about the *types* of the expressions. By the end of lecture, you should know

- what the word “semantics” means.
- how to structure a proof-tree
- how use the type rules to prove the type of an expression
- write your own type rule for an expression

Strongly Typed Languages

- e.g., ML, Haskell, most of C++, C
- You have to declare all your types at the beginning
- No type checks during run-time (faster, safer code)

Weakly Typed Languages

- e.g., Perl, Lisp, some parts of Java
- Very flexible programming
- Types must be checked at run-time or Bad Things happen

Other advantages of strong typing... §1 Why Type Checking is Good

Engineering

- A program that type-checks is likely to be correct.
- Types constrain the use of a function.

Safety • Every try casting an integer into a pointer in C?

```
1  int i = 5;  
2  *((int *)i) = 10;
```

- A type error in an untyped language can be very hard to detect.

Theory

- It is much easier to verify the operation of type-correct code.

To create a formal system, you must specify the following:

- A set of *symbols* or an *alphabet*.
- A definition of a *valid sentence*.
- A set of *transformation rules* to make new valid sentences out of old ones.
- A set of *initial valid sentences*.

You do **NOT** need:

- An *interpretation* of those symbols.
They are highly recommended, but the formal system can exist and do its work without one.

Symbols $S, (,), Z, P, x, y$.

Definition of a furbitz

- Z is a furbitz. x and y are variables of type furbitz.
- if x is a furbitz, then $S(x)$ is a furbitz.
- if x and y are furbitzi, then $P(x, y)$ is a furbitz.

Definition of the gloppit relation

- Z has the gloppit relation with Z .
- If x and y have the gloppit relation, then $S(x)$ and $S(y)$ have the gloppit relation.
- If α and β , then we can write $\alpha g \beta$.

True Sentences If $\alpha g \beta$, then also

- $P(S(\alpha), \beta) g P(\alpha, S(\beta))$, and $P(Z, \alpha) g \alpha$

Symbols $S, (,), Z, P, x, y$.

Definition of an integer

- 0 is an integer. x and y are variables of type integer.
- if x is an integer, then $S(x)$ is an integer.
- if x and y are integers, then $P(x, y)$ is an integer.

Definition of the equality relation

- 0 has the equality relation with 0.
- If x and y have the equality relation, then $S(x)$ and $S(y)$ have the equality relation.
- If α and β , then we can write $\alpha = \beta$.

True Sentences If $\alpha = \beta$, then also

- $P(S(\alpha), \beta) = P(\alpha, S(\beta))$, and $P(0, \alpha) = \alpha$

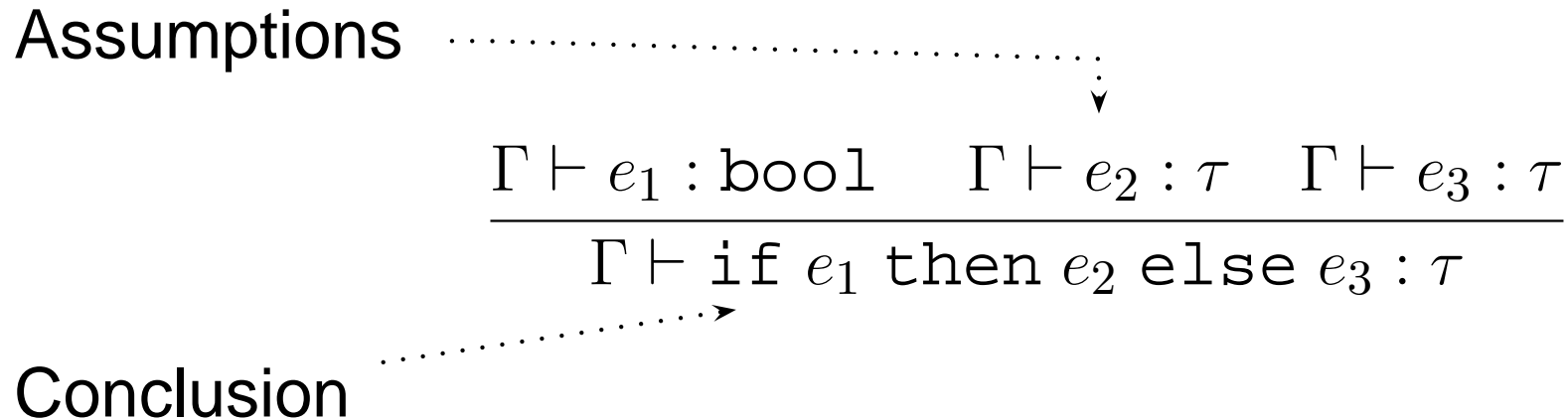
An *type judgment* has the following form:

$$\Gamma \vdash e : \tau$$

where Γ is a *type environment*, e is some expression, and τ is a *type*.

- $\Gamma \vdash \text{if true then 4 else 38} : \text{int}$
- $\Gamma \vdash \text{true \&\& false} : \text{bool}$

Note: the \vdash is pronounced “turnstile” or “entails”.



- If a rule has no assumptions, then it is called an *axiom*.
- Γ is a list of the form $[x : \tau; \dots]$.
- Γ may be left out if we don't need a type environment (see next slide).
- **Basic Idea:** The meaning of an expression can be determined by combining the meaning of its parts.

Constants $\frac{}{\vdash n : \text{int}}$ (assuming n is an int)

$$\frac{}{\vdash \text{true} : \text{bool}}$$
$$\frac{}{\vdash \text{false} : \text{bool}}$$

Variables $\frac{}{\Gamma \vdash x : \tau}$, if $x : \tau \in \Gamma$

- The Variable Rule is actually a bit more complicated in real life, but this form is sufficient.
- These are rules that are true no matter what the context is.

Arithmetic

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \oplus e_2 : \text{int}}$$

Relations

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \sim e_2 : \text{bool}}$$

Booleans

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \&\& e_2 : \text{bool}}$$

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \mid \mid e_2 : \text{bool}}$$

- These are combination rules.

Suppose we want to prove that $\Gamma \vdash (x * 5 > 7) \&\& y : \text{bool}$.

Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

First thing: Write down the thing you are trying to prove, and put a bar over it.

$$\overline{\Gamma \vdash (x * 5 > 7) \&\& y : \text{bool}}$$

Look at the *outermost* expression. What kind of expression is this?

Use the rule
$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \&\& e_2 : \text{bool}}$$

You need the “and” rule. It will tell you which parts of the goal need to be proved next.

Suppose we want to prove that $\Gamma \vdash (x * 5 > 7) \&\& y : \text{bool}$.
Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

Write parts on top and put a bar over them as well.

$$\frac{\overline{\Gamma \vdash x * 5 > 7 : \text{bool}} \quad \overline{\Gamma \vdash y : \text{bool}}}{\Gamma \vdash (x * 5 > 7) \&\& y : \text{bool}}$$

What to do next? Let's work left to right. The expression we want next is a “greater” expression. (Besides, the y expression is already an axiom.)

Suppose we want to prove that $\Gamma \vdash (x * 5 > 7) \&\& y : \text{bool}$.
Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

Following the “greater” rule, we break the $x * 5 > 7$ into two parts.

$$\frac{\frac{\overline{\Gamma \vdash x * 5 : \text{int}} \quad \overline{\Gamma \vdash 7 : \text{int}}}{\Gamma \vdash x * 5 > 7 : \text{bool}} \quad \overline{\Gamma \vdash y : \text{bool}}}{\Gamma \vdash (x * 5 > 7) \&\& y : \text{bool}}$$

We will turn our attention to the multiplication now.

Suppose we want to prove that $\Gamma \vdash (x * 5 > 7) \&\& y : \text{bool}$.
 Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

At this point, there are no more subtrees to expand out. We are done.

$$\begin{array}{c}
 \frac{}{\Gamma \vdash x : \text{int}} \quad \frac{}{\Gamma \vdash 5 : \text{int}} \\
 \hline
 \Gamma \vdash x * 5 : \text{int} \quad \frac{}{\Gamma \vdash 7 : \text{int}} \\
 \hline
 \Gamma \vdash x * 5 > 7 : \text{bool} \quad \frac{}{\Gamma \vdash y : \text{bool}} \\
 \hline
 \Gamma \vdash (x * 5 > 7) \&\& y : \text{bool}
 \end{array}$$

$$\text{If } \frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : \tau}$$

- The τ means “any type at all”—but whatever type τ you pick it has to be the same for the three places it shows up in this rule.
- So... the `if` rule says that `if` can result in any type, as long as the `then` and `else` branches have the same type. This could even include functions.

$$\frac{\Gamma \vdash e : \tau_1 \rightarrow \tau_2 \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau \quad \Gamma \vdash e_1 : \tau_1 \quad \cdots \quad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash e \ e_1 \ e_2 \ \cdots \ e_n : \tau}$$

- If you have a function of type $\tau_1 \rightarrow \tau_2 \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau$, and if every argument e_i has type τ_i , then applying them **in that order** will produce an expression of type τ .

$$\frac{\Gamma \vdash \text{map} : (\alpha \rightarrow \beta) \rightarrow \alpha \text{ list} \rightarrow \beta \text{ list} \quad \Gamma \vdash f : (\alpha \rightarrow \beta) \quad \Gamma \vdash \text{lst} : \alpha \text{ list}}{\Gamma \vdash \text{map } f \text{ lst} : \beta \text{ list}}$$

- For “compound types” like $\alpha \text{ list}$, we only substitute in the α parts.

- Important point: the rules describe types, but they also describe when you may change Γ .
- You may **NOT** change Γ except as described!

$$\textbf{Functions} \quad \frac{\Gamma \cup [x_1 : \tau_1; \dots; x_n : \tau_n] \vdash e : \tau}{\Gamma \vdash \text{fun } x_1 \dots x_n \rightarrow e : \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau}$$

$$\textbf{Let} \quad \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \cup [x : \tau] \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'}$$

$$\textbf{Let Rec} \quad \frac{\Gamma \cup [x : \tau] \vdash e_1 : \tau \quad \Gamma \cup [x : \tau] \vdash e_2 : \tau'}{\Gamma \vdash \text{let rec } x = e_1 \text{ in } e_2 : \tau'}$$

Type-checking compose:

$$\frac{}{\Gamma \vdash \text{fun } f \text{ } g \text{ } x \rightarrow f \text{ } (g \text{ } x) : (\alpha \rightarrow \beta) \rightarrow (\delta \rightarrow \alpha) \rightarrow \delta \rightarrow \beta}$$

Before using the function rule:

$$\frac{\Gamma \cup [x_1 : \tau_1; \dots; x_n : \tau_n] \vdash e : \tau}{\Gamma \vdash \text{fun } x_1 \dots x_n \rightarrow e : \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau}$$

Type-checking compose:

$$\frac{\overline{\Gamma' = \Gamma \cup \{f : (\alpha \rightarrow \beta); g : (\delta \rightarrow \alpha); x : \delta\} \vdash f (g x) : \beta}}{\Gamma \vdash \text{fun } f \text{ } g \text{ } x \rightarrow f (g x) : (\alpha \rightarrow \beta) \rightarrow (\delta \rightarrow \alpha) \rightarrow \delta \rightarrow \beta}$$

After using the function rule:

$$\frac{\Gamma \cup [x_1 : \tau_1; \dots; x_n : \tau_n] \vdash e : \tau}{\Gamma \vdash \text{fun } x_1 \dots x_n \rightarrow e : \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau}$$

Next, we use application.

$$\frac{\Gamma \vdash e : \tau_1 \rightarrow \tau_2 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau \quad \Gamma \vdash e_1 : \tau_1 \quad \dots \quad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash e \ e_1 \ e_2 \ \dots \ e_n : \tau}$$

Type-checking compose:

$$\frac{\frac{\overline{\Gamma' \vdash f : (\alpha \rightarrow \beta)} \quad \overline{\Gamma' \vdash (g \ x) : (\alpha)}}{\Gamma' = \Gamma \cup \{f : (\alpha \rightarrow \beta); g : (\delta \rightarrow \alpha); x : \delta\} \vdash f \ (g \ x) : \beta}}{\Gamma \vdash \text{fun } f \ g \ x \rightarrow f \ (g \ x) : (\alpha \rightarrow \beta) \rightarrow (\delta \rightarrow \alpha) \rightarrow \delta \rightarrow \beta}$$

The rule for parens is trivial... can you write it?

Next, we'll use application again.

Type-checking compose:

$$\frac{
 \frac{
 \overline{\Gamma' \vdash f : (\alpha \rightarrow \beta)}
 \quad
 \frac{
 \overline{\Gamma' \vdash g : (\delta \rightarrow \alpha)}
 \quad
 \overline{\Gamma' \vdash x : \delta}
 }{
 \overline{\Gamma' \vdash (g \ x) : \alpha}
 }
 }{
 \overline{\Gamma' = \Gamma \cup \{f : (\alpha \rightarrow \beta); g : (\delta \rightarrow \alpha); x : \delta\} \vdash f \ (g \ x) : \beta}
 }
 }{
 \Gamma \vdash \text{fun } f \ g \ x \rightarrow f \ (g \ x) : (\alpha \rightarrow \beta) \rightarrow (\delta \rightarrow \alpha) \rightarrow \delta \rightarrow \beta
 }$$

We are done. These rules are meant mainly to *verify* a type, but they can be used to *infer* a type as well.

At first, give everything a separate type.

$$\frac{}{\Gamma \vdash \text{fun } f \text{ } g \text{ } x \rightarrow f (g \text{ } x) : \alpha \rightarrow \beta \rightarrow \delta \rightarrow \gamma}$$

Apply the function rule:

$$\frac{\Gamma' = \Gamma \cup \{f : \alpha; g : \beta; x : \delta\} \vdash f (g x) : \gamma}{\Gamma \vdash \text{fun } f \text{ } g \text{ } x \rightarrow f (g x) : \alpha \rightarrow \beta \rightarrow \delta \rightarrow \gamma}$$

Apply the application rule:

$$\frac{\frac{\overline{\Gamma' \vdash f : \alpha} \quad \overline{\Gamma' \vdash (g \ x) : ???}}{\Gamma' = \Gamma \cup \{f : \alpha; g : \beta; x : \delta\} \vdash f (g \ x) : \gamma}}{\Gamma \vdash \text{fun } f \ g \ x \rightarrow f (g \ x) : \alpha \rightarrow \beta \rightarrow \delta \rightarrow \gamma}$$

From here we see that α needs to be a function type. Also, we need to decide a type for $(g \ x)$.

- Let $(g \ x)$ have type ν .
- Let f have type $\alpha = \nu \rightarrow \gamma$.

We make the appropriate substitutions....

Apply the application rule:

$$\frac{\frac{\overline{\Gamma' \vdash f : (\nu \rightarrow \gamma)} \quad \overline{\Gamma' \vdash (g \ x) : \nu}}{\Gamma' = \Gamma \cup \{f : (\nu \rightarrow \gamma); g : \beta; x : \delta\} \vdash f \ (g \ x) : \gamma}}{\Gamma \vdash \text{fun } f \ g \ x \rightarrow f \ (g \ x) : (\nu \rightarrow \gamma) \rightarrow \beta \rightarrow \delta \rightarrow \gamma}$$

Now we use the application rule for $(g \ x)$.

Apply the application rule:

$$\frac{
 \frac{
 \overline{\Gamma' \vdash f : (\nu \rightarrow \gamma)}
 }{
 \overline{\Gamma' = \Gamma \cup \{f : (\nu \rightarrow \gamma); g : \beta; x : \delta\} \vdash f (g x) : \gamma}
 }{
 \Gamma \vdash \text{fun } f \text{ } g \text{ } x \rightarrow f (g x) : (\nu \rightarrow \gamma) \rightarrow \beta \rightarrow \delta \rightarrow \gamma
 }$$

● We know that β needs to be $\delta \rightarrow \nu$, because of our rule.

Substituting, we get...

Apply the application rule:

$$\frac{\displaystyle \frac{\Gamma' \vdash f : (\nu \rightarrow \gamma)}{\Gamma' = \Gamma \cup \{f : (\nu \rightarrow \gamma); g : (\delta \rightarrow \nu); x : \delta\} \vdash f (g x) : \gamma} \quad \displaystyle \frac{\Gamma' \vdash g : (\delta \rightarrow \nu) \quad \Gamma' \vdash x : \delta}{\Gamma' \vdash (g x) : \nu}}{\Gamma \vdash \text{fun } f \ g \ x \rightarrow f (g x) : (\nu \rightarrow \gamma) \rightarrow (\delta \rightarrow \nu) \rightarrow \delta \rightarrow \gamma}$$

Now we're done.

Here's an example I showed one time I gave this lecture.

$$\frac{
 \frac{
 \frac{}{\Gamma \cup [x : \tau] \vdash x : \tau}
 }{
 \Gamma \vdash \text{fun } x \rightarrow x : \tau \rightarrow \tau
 }
 \quad
 \frac{
 \frac{}{\Gamma^1 \vdash \text{id} : \tau \rightarrow \tau}
 }{
 \Gamma^1 \vdash \text{id} (\text{id } 10) : \text{int}
 }
 \quad
 \frac{
 \frac{}{\Gamma^1 \vdash \text{id} : \tau \rightarrow \tau} \quad \frac{}{\Gamma^1 \vdash 10 : \text{int}}
 }{
 \Gamma^1 \vdash (\text{id } 10) : \text{int}
 }
 }{
 \Gamma \vdash \text{let id} = \text{fun } x \rightarrow x \text{ in id } (\text{id } 10) : \text{int}
 }$$

- Let $\Gamma^1 = \Gamma \cup [\text{id} : \tau \rightarrow \tau]$
- What would be different if we'd used `let rec` instead?

Try these problems.

- Prove that $\Gamma \vdash \text{let } f = \text{fun } x \rightarrow x + 2 \text{ in let } g = \text{fun } x \rightarrow x + 3 \text{ in (if } 4 > 6 \text{ then } f \text{ else } g) 10 : \text{int}$
- Write a type judgment rule for the list operator $::$.